Kang and Choo *EURASIP Journal on Wireless Communications and Networking* (2016) 2016:25
DOI 10.1186/s13638-016-0523-6

EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

A cluster-based decentralized job dispatching for the large-scale cloud

Byungseok Kang* and Hyunseung Choo

Abstract

The remarkable development of cloud computing in the past few years, and its proven ability to handle web hosting workloads, is prompting researchers to investigate whether clouds are suitable to run large-scale computations. Cloud load balancing is one of the solution to provide reliable and scalable cloud services. Especially, load balancing for the multimedia streaming requires dynamic and real-time load balancing strategies. With this context, this paper aims to propose an Inter Cloud Manager (ICM) job dispatching algorithm for the large-scale cloud environment. ICM mainly performs two tasks: clustering (neighboring) and decision-making. For clustering, ICM uses Hello packets that observe and collect data from its neighbor nodes, and decision-making is based on both the measured execution time and network delay in forwarding the jobs and receiving the result of the execution. We then run experiments on a large-scale laboratory test-bed to evaluate the performance of ICM, and compare it with well-known decentralized algorithms such as Ant Colony, Workload and Client Aware Policy (WCAP), and the Honey-Bee Foraging Algorithm (HFA). Measurements focus in particular on the observed total average response time including network delay in congested environments. The experimental results show that for most cases, ICM is better at avoiding system saturation under the heavy load.

Keywords: Cluster-based, Cloud computing, Inter Cloud Manager, Job dispatching

1 Introduction

In the past two decades, cloud computing has emerged as an enabling technology and it has been increasingly adopted in many areas including business, science, and engineering because of its inherent scalability, flexibility, and cost-effectiveness [1]. Currently, cloud computing is providing dynamic services like applications, data, hardware resources, and various IT services over the internet. The reliability and performance of cloud services depend on various factors including load balancing [2, 3] and job dispatching [4, 5]. Job dispatching is performed on the basis of different parameters so that it increases the overall cloud performance. A job may include entering data, processing, accessing software, or storage functions. The data center classifies jobs according to the service-level agreement and requested services. Each job is then assigned to one of the available servers. In turn, the servers perform the requested job, and a response is transmitted back to the user. With this requirement, current cloud data center needs to support large-scale global coverage [6], low cost [7, 8], low latency [9, 10], certain level of security [11, 12], and application availability for the customers.

Scalability in cloud is one of the major advantages brought by the cloud paradigm. However, there is an important issue of job dispatching among heterogeneous clouds. Efficient use of computing resources to minimize the execution time requires a job dispatching algorithm that can appropriately determine the assignment of jobs. Due to this variety of jobs and servers, task scheduling mechanisms for single system may not be effective in a distributed environment [13–15]. Furthermore, mobile devices [16] are becoming one of the major sources of the workload for clouds in order to save energy at the mobile device itself and avoid moving bulky data over wireless networks between the mobile devices and data repositories.

However, it should be noted that the job dispatching itself is another kind of system overhead [17, 18]. These overheads include the following: the time required by computing hosts/nodes for updating their system load

*Correspondence: byungseok@skku.edu
Department of Computer Science and Engineering, Sungkyunkwan University, Suwon 440-746, South Korea

information in a real time manner, the communication costs for sharing those load information to make a decision, and the costs of job transmission. Therefore, the conditions of job dispatching is worth investigating and how they work should be considered. In order to handle different users' job requests, a job dispatcher/controller monitors the large-scale and heterogeneous cloud systems with low cost (overhead).

Although the parallelization strategy enables scalability [19], a good load balancing scheme is necessary to achieve good performance. In this paper, we introduced an Inter Cloud Manager (ICM) job dispatching algorithm which is operating in not only small scale (centralized) but also large-scale (decentralized) environments. We compared its performance with three state-of-the-art load balancing algorithms. In the results, ICM showed superior performance of average response time in the congested situation. It means that the proposed ICM provides scalability based on clustering and decision-making. In addition, by using this experimental results, we can design the appropriate number of cloud server resources while changing the system loads. The rest of this paper is organized as follows. A literature review on decentralized load balancing algorithms in clouds and related technologies are presented in Section 2. In Section 3, we detail our proposed ICM. Experiments and measurement results are provided in Section 4. Finally, we conclude the paper in Section 5.

2 Related works

Load balancing has become an attractive issue since the emergence of distributed systems. Load balancing algorithms can be classified into sub categories from various perspectives. From one point of view, they can be classified into centralized and decentralized algorithms. The case where the load balancer resides at the master node is called centralized load balancing policy, while the other case where the load balancer resides at all the nodes under consideration is called the distributed (decentralized) load balancing policy. In this section, we mainly discuss the mostly known contributions in decentralized algorithms for the large-scale cloud.

In [20], an Ant Colony Optimization technique that improves upon the work in [21] was suggested. Both algorithms are using the ants' behavior to gather information about the cloud hosts to assign the task to a specific host. However, the algorithm in [21] has a problem with ant synchronization and the author in [20] tried to solve this by adding the feature "suicide" to the ants. The Ant Colony algorithm has many advantages compared to other static algorithms. The advantages include fast decision-making, no single point of failure (SPOF), and low complexity.

In [22], a map-reduce-based entity resolution approach was discussed. It has two main tasks: *map* and *reduce* tasks. Since several *map* tasks can read entities in parallel and process them, the *reduce* adds one more load balancing level between the *map* task and the *reduce* task for the purpose of decreasing the overload on these tasks. The job dispatching in the middle stage divides only the large tasks into smaller tasks and then these smaller tasks are sent to the *reduce* tasks based on their availability.

In [23], a dual direction downloading algorithm from FTP servers (DDFTP) that is available over the Internet, cloud, and grid environments was proposed. This technique utilizes the availability of replicated FTP servers to enhance file download times through concurrent downloads of file blocks. The algorithm reduces the network communication needed between the clients and hosts and therefore reduces the network overhead. Most of the distributed algorithms take quite a long time for their decision-making process. However, DDFTP has low complexity and dispatches a job very fast.

The algorithm proposed in [24] is a load balancing for Internet distributed services (IDS) which are distributed all over the world. A middleware is described to implement this protocol. IDS also uses a heuristic to help web servers to endure overloads. It reduces the service response times that limits the redirection of requests to the closest remote servers without overloading them. IDS is a complex algorithm and generates a large amount of network overhead, but its decision-making process is fast.

In [25], a dynamic load balancing algorithm called load balancing min-min (LBMM) technique is presented which is based on three level frameworks. LBMM helps in an efficient utilization of resources and enhances the work efficiency. However, LBMM itself is highly complex and generates a high amount of additional dummy packets. Furthermore, the distributed LBMM algorithm takes quite a long time for the decision-making process (algorithm speed).

The paper [26] proposed a new content-aware load balancing policy named as Workload and Client Aware Policy (WCAP). It applies a technique to specify the Unique and Special Property (USP) of the requests as well as computing nodes. Based on USP, the scheduler decides the node that is best suitable for the processing of the requests. This strategy is implemented in a decentralized manner with high overhead. By using the content information to narrow down the search, this technique improves the searching performance and hence the overall performance of the system. It also helps in reducing the idle time of the computing nodes, hence improving their utilization.

In [27], a honey bee behavior-inspired load balancing algorithm for cloud environment was proposed. This algorithm is derived from the behavior of honey bees in finding their food. Among the classes of bees, the forager bees forage for food sources. In case of load balancing, the servers are grouped into a virtual server (VS). Each

Table 1 Description of Hello packet field

Field name	Bytes	Description
Source address	4	The IP address of the host is sending to
Sequence number	2	Unique sequence number of this message
Max. hop count	2	The number of maximum hop count this message travel the networks
Hello interval	2	The number of seconds source node waits between sending Hello messages
Dead interval	2	The number of seconds a router can be 'silent' before it is considered to have failed

VS calculates its profit which is similar to the bees' waggle dance.

3 Inter Cloud Manager

In this section, we introduce and describe an Inter Cloud Manager (ICM) that is designed for the large-scale cloud. ICM consists of two main parts: clustering and decision-making.

3.1 Clustering

Data centers for cloud computing continue to grow in terms of both hardware resources and traffic volume, thus making cloud operation and management more and more complex. In this scenario, accurate and fine-grained monitoring [28, 29] activities are required to efficiently operate these platforms and to manage their increasing complexity. Furthermore, in order to be able to meet demands and provide satisfactory QoS [30–33], individual monitoring mechanisms are needed and can lead to collection and processing of a large amount of runtime data. To monitor the clouds continuously, we use "Hello" packet to collect system load and the end-to-end delay from the client to the host. A Hello packet is sent periodically on each network interface to discover and test connections among neighbors. Hello packets are broadcasting to enable dynamic router and host server discovery. The field structure of Hello packet is shown in Table 1 and Fig. 1 in the body of the message. Especially, max. hop count is a key parameter in our measurement because this parameter determined the boundary of neighbors.

The ACK packet is sent by the receiving server (destination) and goes back to the sending server (source). Every interconnected server information is periodically updated by Hello packet through ACK packet that is being sent by the destination server. When the source receives the ACK packet from the destination, it computes and stores network link (hop count, delay, loss) and system load (job execution time, memory usage, number of waiting jobs) information into its neighbor table. Each cloud host can make its own "cluster" based on this table. The details of the ACK packet is described in both Table 2 and Fig. 2.

3.2 Decision-making

Decision-making is a core function of ICM and maintains best efficiency in dispatching job requests from a client in the ideal cloud location. Algorithm 1 shows the details of ICM's decision-making process. This decision-making process is triggered when the system's number of waiting jobs in the waiting queue is larger than 5. There are two steps that decision-making follows.

The first step is selecting one request among the job list that ICM holds. All job requests are saved in a "linked list" containing its first time that it was originally entered as job request and a basic description. One reason to use the linked list is to provide primary benefit of limiting memory waste as insertion, and removal of data constantly takes place. ICM processes job requests in order, from head to tail which means in our case that we used a first-in-first-out (FIFO) job selection process. We will explain details about experimental system settings in the following section.

The second step is the job dispatching process that was initially placed by the client and its task is to make sure that the dispatching process is well undertaken to the most ideal cloud host. ICM uses a "best-fit" approach that dispatches a job to the shortest average response time (ART) among operating clouds. Firstly, ICM checks the ART value. If all the hosts exceed the threshold of ART, ICM does not send and holds a job until a host's ART is under the threshold. Secondly, ICM checks the status of hosts to see whether they are idle or not. If a host does not have any waiting jobs, ICM sends a job to that host. Finally, ICM sends a job to the host with the minimum value of ART.

As we discussed, ICM uses history base decision-making. We can obtain the network link and system load informations from the Hello-ACK packet. In this case, ART is calculated by summing between expected

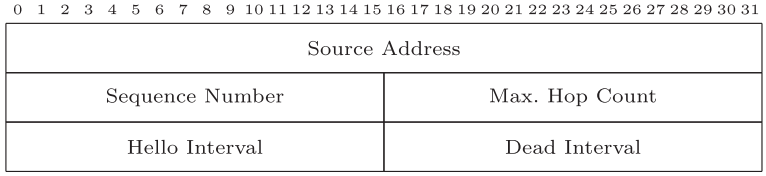

Fig. 1 Hello packet format

Table 2 Description of ACK packet field

Field name	Bytes	Description
ACK source address	4	The address of the node originating the acknowledgement
ACK destination address	4	The address of the node to which the acknowledgement is to be delivered
Execution time	2	The execution time of recently executed (finished) two jobs
Memory usage	2	The memory usage information of recently executed two jobs
Num. of waiting jobs	2	The number of jobs in the waiting queue
Options	2	Variable-length field

network delay (ENT) and expected job transfer time (EJTT) from the cloud host. ENT is calculated by using the formula (avg. network delay × num. of packets for current job) over (1 – loss rate). From a network's point of view, a job (application) consists of several numbers of data packets. In our case, avg. network delay included processing, queuing, transmission, and propagation delays. EJTT is calculated by avg. job execution time × (num. of waiting jobs +1). Average job execution time can be calculated as shown in expression (1) where W_{new} stands for current average job execution time and W_{old} is previous value, W_{last} is last job execution time. a is the value ($0 < a < 1$) that is worthwhile to notice as it is the usage of system memory from the last job within the cloud host. For instance, suppose the latest job used 30 % of system memory. Therefore, the value of a becomes 0.3 in the simplest term.

$$W_{\text{new}} \leftarrow W_{\text{old}}(1-a) + W_{\text{last}} \times a \tag{1}$$

Algorithm 1 Decision-making of ICM

Ensure: Number of waiting jobs in system > 5
  Select job(j)
  Compute each neighbor's observed Average Response Time (ART)
  /*no appropriate neighbor at current time*/
  if Minimum ART ≥ threshold(ART) **then**
    Do not Dispatch Job(j)
    /*prevent starvation of the Cloud*/
  else if neighbor[N_ID] has no waiting jobs **then**
    Allocate job(j) to the neighbor[N_ID]
  else
    Allocate job(j) to the appropriate neighbor
  end if

4 Experimental results

We have implemented experiment environments on a cloud test-bed with different types of server machines. We use a total of 15 servers for the experiment; 5 host servers have Intel Xeon 2.4 GHz dual core CPUs, and size of RAM is 0.5 GB where 5 have Xeon 3.0 GHz dual core CPUs with RAM size of 2.0 GB. Finally, the n five hosts have Xeon 2.8 GHz dual core CPUs with a RAM size of 1.0 GB. Each server physically distributes and runs Linux (Ubuntu) with CPU throttling enabled with the *on demand governor*, which dynamically adjusts the cores frequencies depending on load. In all experiments, we conducted both averaged measurements for long periods and observed the job response time while changing the job arrival rate.

4.1 System settings and job processing

As shown in Fig. 3, the system setting is largely divided into four client mobile devices and 15 cloud host servers. We use a Google reference mobile phone for the client machine. The client sends 32×32 matrix inversion jobs through the WiFi protocol. This 32×32 matrix is randomly generated by the client job generator. The client just sends a fraction of job requests with matrix data to the destination. The ICM module (controller) is located inside every server and follows the rule. The controller plays an important role not only in interconnecting the client with cloud hosts but in dispatching jobs to the ideal cloud host. The controller uses four different algorithms for dispatching a job, such as proposed ICM, Ant Colony, HFA, and WCAP.

A job is executed when the request arrives from the controller. After receiving this request, the system runs binary code and sends the result back to the client over the network. Each host server has different hardware resources (e.g., CPU, memory, storage) and network propagation delay values (10–5000 ms). In addition, each server processes jobs based on first-come-first-served (FCFS) scheduling policy. That is the commonly used and simplest way to develop a single waiting queue. Below are several steps of establishing a network connection among the client, controller, and the host. That is the basics of our cluster-based job dispatching.

1. The client attempts to connect with the job on the controller (SYN).
2. The controller accepts the connection, and after deciding which host should receive the connection, changes the destination IP (and possibly port) to match the job of the selected host (note that the source IP of the client is not touched).
3. The host accepts the connection and responds back to the original source, the client, via its default route, the controller (SYN/ACK).

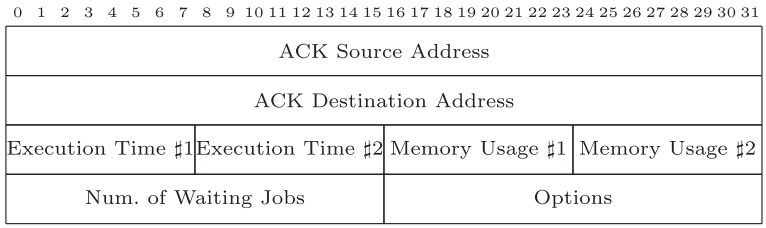

Fig. 2 ACK packet format

4. The controller intercepts the return packet from the host and now changes the source IP (and possible port) to match the controller IP and port and forwards the packet back to the client.
5. The client receives the return packet, believing that it came from the controller, and establishes a network session (ACK)
6. The controller receives a job request packet from the client and forwards the packet to the appropriate host (decision-making)

4.2 Measurement results

We evaluate the performance of the proposed algorithm on a cloud test-bed and compare it with three decentralized algorithms (Ant Colony [20], WCAP [26], HFA [27]). We fixed the threshold value of ART for 3 min at every measurement. If every system has reached that value, ICM does not dispatch a job to the cloud hosts anymore. We also fixed the dispatching trigger point of each host as 5. It means that if the number of waiting jobs in a system is over 5, ICM starts forwarding remaining jobs to its neighbor.

The results for each algorithm are shown in Figs. 4, 5, and 6. The reported values were obtained by averaging the measurements. We changed the job arrival rate λ from 0.1 to 0.9 and measured 30 times at each λ point. For instance, if the λ is 0.5, all four clients send job requests with 0.5 requests per second in a probabilistic way. Figure 4 highlights the relative performance of the four algorithms while changing the max. hop count parameter from 2 to 4. In that measurement, we fixed the parameter of hello time interval for 10 s. In each case, ICM shows outstanding performance compared to other load balancing algorithms. ICM has smooth curves while others are rapidly increased for high traffic rates. One interesting thing is that max. hop count $= 3$ shows the best performance. If the max. hop count of Hello packet is larger than 3, it causes degradation of system performance.

In the case of Fig. 5, we changed hello interval parameter from 10 to 30 s. and fixed max. hop count $= 3$. While increasing the hello time interval, ICM's performance slowly decreased. Four algorithms show stable performance when the traffic rate λ is lower than 0.5. But, three algorithms reached system saturation when the traffic rate is higher than 0.6 while ICM still works well until λ reaches 0.7.

The four algorithms impose vastly different amounts of overhead, as shown in Fig. 6. We changed only the hello time interval parameter (10–30 s). As the results, WCAP generate least control message overhead. The proposed ICM mostly generate additional message at hello time interval $= 10$, but when hello interval is larger than 20, the amount of overhead is less than the Ant Colony algorithm. Ant Colony uses the ants' behavior to collect information of cloud node. However, it could easily cause a network overhead due to the large number of dispatched ants.

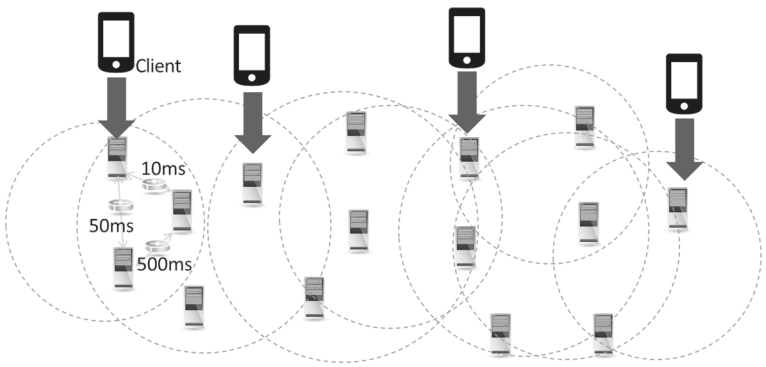

Fig. 3 System settings

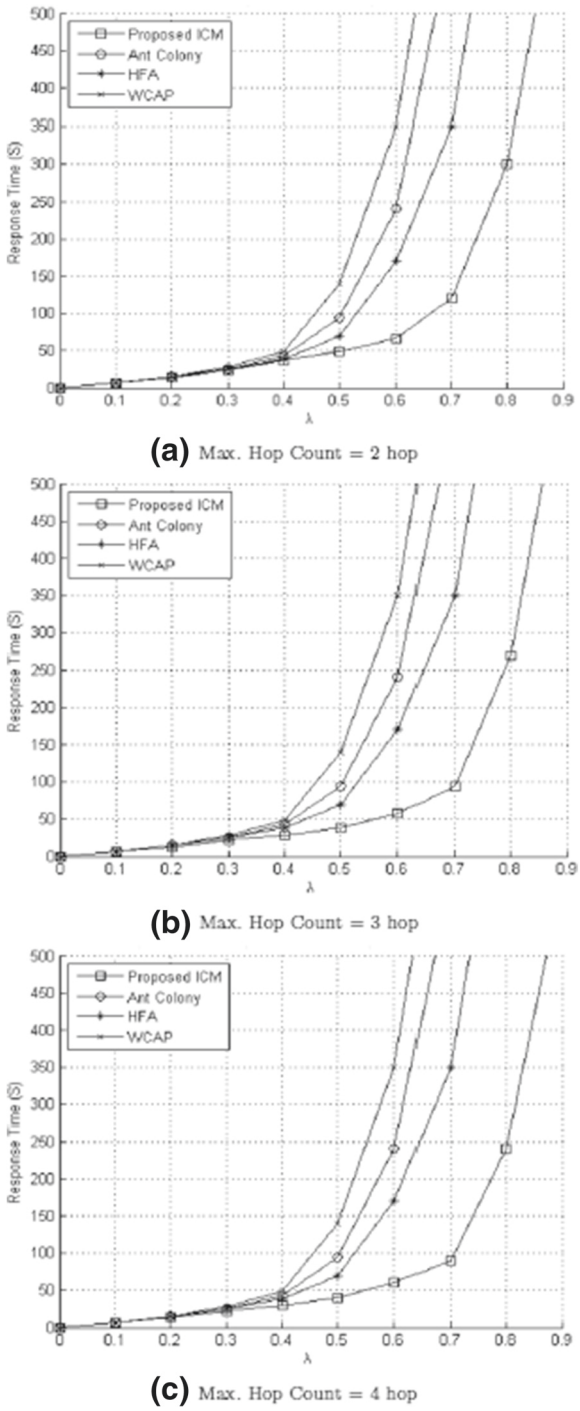

Fig. 4 Measured response time while changing max. hop count

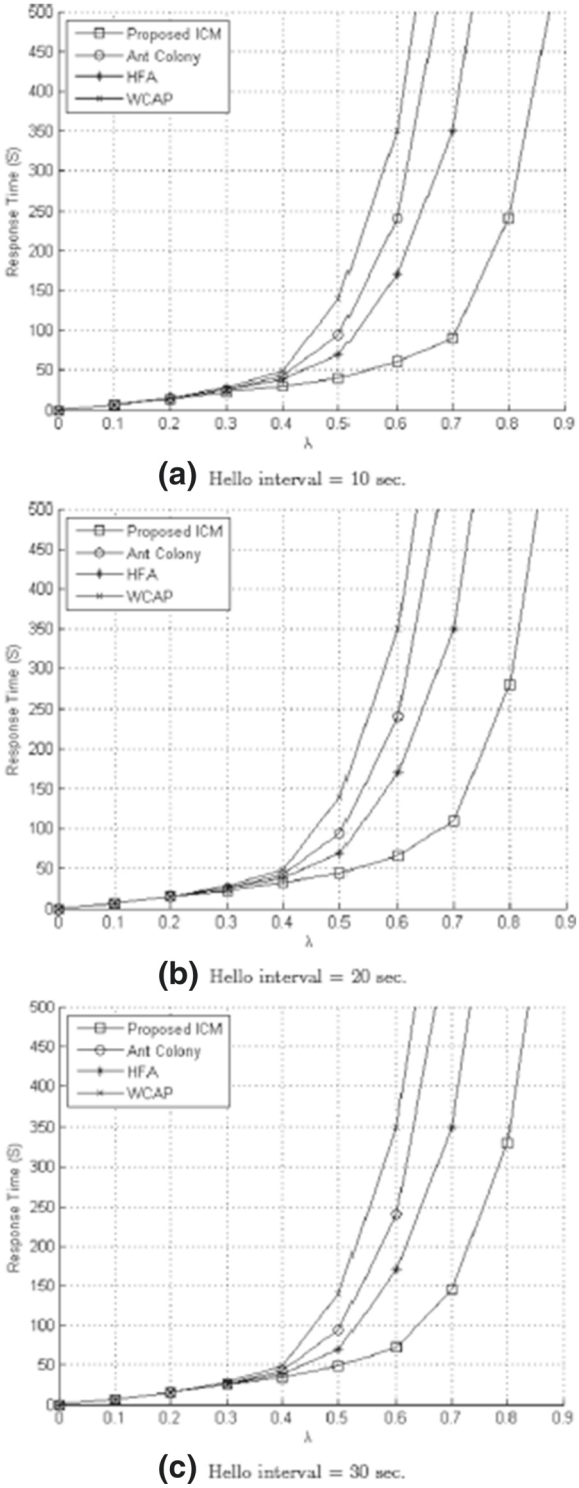

Fig. 5 Measured response time while changing Hello interval

5 Conclusions

In this paper, we have proposed a decentralized job dispatching algorithm, designed to be suitable for the large-scale cloud environment. The proposed ICM uses additional Hello packets that observe and collect data. Comparative experimental measurement is carried out to compare the performance of ICM, Ant Colony, WCAP, and HFA while increasing the job sending rate. After evaluation, average response time from ICM demonstrated

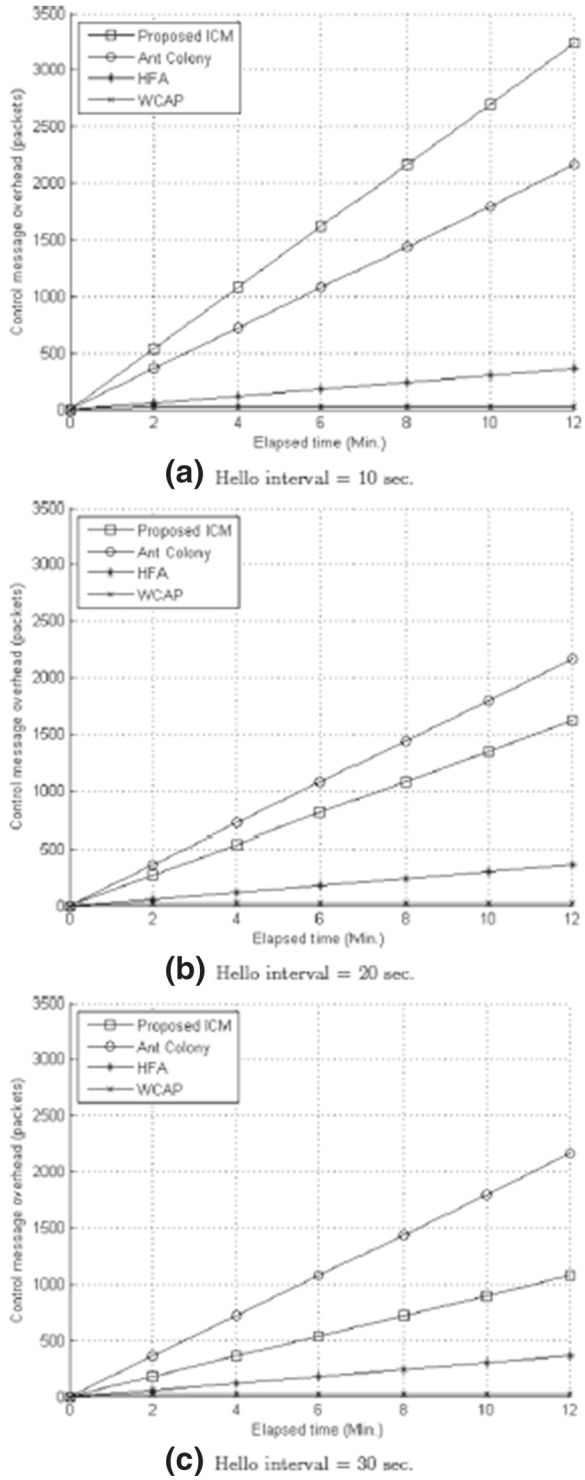

Fig. 6 Control message overhead while changing Hello interval

a higher performance than the other three algorithms. To use these experimental results, we can estimate the expected saturation point in cloud systems. However, in our system experiment, the client just sent a fraction of the computation job request to the destination. But in the real environment [34–36], congestion can occur in any intermediate node, often due to limitation in resources, when data packets are being transmitted from the client to the destination. Congestion will lead to high packet loss, long delay, and a waste of resource utilization time. Therefore, in our next work, we will use communication jobs such as video streaming and VoIP rather than computation jobs.

Competing interests
The authors declare that they have no competing interests.

Acknowledgements
This work was supported by Priority Research Centers Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education, Science and Technology(2010-0020210) and the MSIP, Korea, under the G-ITRC support program (IITP-2015-R6812-15- 0001) supervised by the IITP.

Received: 22 November 2015 Accepted: 12 January 2016
Published online: 20 January 2016

References

1. MR Rahimi, J Ren, CH Liu, AV Vasilakos, N Venkatasubramanian, Mobile cloud computing: a survey, state of art and future directions. Mobile Netw. Appl. **19**(2), 133–143 (2014)
2. M Arshad, S Malik, An efficient algorithm for load balancing in cloud computing. Futur. Gener. Comput. Syst (2014)
3. M Katyal, A Mishra, *A comparative study of load balancing algorithms in cloud computing environment*, (2014). arXiv preprint arXiv:14036918
4. W Zhang, Y Wen, J Cai, DO Wu, Toward transcoding as a service in a multimedia cloud: energy-efficient job-dispatching algorithm. IEEE Trans. Veh. Technol. **63**(5), 2002–2012 (2014)
5. K Dutta, RB Guin, S Chakrabarti, S Banerjee, U Biswas. A smart job scheduling system for cloud computing service providers and users: modeling and simulation, in *Recent Advances in Information Technology (RAIT), 2012 1st International Conference on,* (IEEE, 2012), pp. 346–351
6. M Polverini, A Cianfrani, S Ren, AV Vasilakos, Thermal-aware scheduling of batch jobs in geographically distributed data centers. IEEE Trans. Cloud Comput. **2**(1), 71–84 (2014)
7. L Wang, F Zhang, K Zheng, AV Vasilakos, S Ren, Z Liu. Energy-Efficient Flow Scheduling and Routing with Hard Deadlines in Data Center Networks, in *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on,* (IEEE, 2014), pp. 248–257
8. L Wang, F Zhang, J Arjona Aroca, AV Vasilakos, K Zheng, C Hou, D Li, Z Liu, Greendcn: A general framework for achieving energy efficiency in data center networks. IEEE J. Sel. Areas Commun. **32**(1), 4–15 (2014)
9. K Chen, C Hu, X Zhang, K Zheng, Y Chen, AV Vasilakos, Survey on routing in data centers: insights and future directions. IEEE Netw. **25**(4), 6–10 (2011)
10. B Wang, Z Qi, R Ma, H Guan, AV Vasilakos, A survey on data center networking for cloud computing. Comput. Netw. **91**, 528–547 (2015)
11. L Wei, H Zhu, Z Cao, W Jia, AV Vasilakos. Seccloud: Bridging secure storage and computation in cloud, in *Distributed Computing Systems Workshops (ICDCSW), 2010 IEEE 30th International Conference on,* (IEEE, 2010), pp. 52–61
12. L Wei, H Zhu, Z Cao, X Dong, W Jia, Y Chen, AV Vasilakos, Security and privacy for storage and computation in cloud computing. Inf. Sci. **258**, 371–386 (2014)
13. E Gelenbe, R Lent, Energy–QoS trade-offs in mobile service selection. Futur. Internet. **5**(2), 128–139 (2013)
14. E Gelenbe, R Lent, M Douratsos. Choosing a local or remote cloud, in *Network Cloud Computing and Applications (NCCA), 2012 Second Symposium on,* (IEEE, 2012), pp. 25–30
15. Sustainable Internet and ICT for Sustainability, SustainIT 2012, 4-5 October, 2012, Pisa, Italy, Sponsored by the IFIP TC6 WG 6.3 Performance of Communication Systems, IEEE (2012). http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=6375585. Access date: January 18, 2016
16. N Fernando, SW Loke, W Rahayu, Mobile cloud computing: a survey. Futur. Gener. Comput. Syst. **29**(1), 84–106 (2013)

17. L Mashayekhy, MM Nejad, D Grosu, A Vasilakos, *An online mechanism for resource allocation and pricing in clouds*, (2015)
18. F Xu, F Liu, H Jin, AV Vasilakos, Managing performance overhead of virtual machines in cloud computing: a survey, state of the art, and future directions. Proc. IEEE. **102**(1), 11–31 (2014)
19. D Niyato, AV Vasilakos, Z Kun. Resource and revenue sharing with coalition formation of cloud providers: game theoretic approach, in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing,* (IEEE Computer Society, 2011), pp. 215–224
20. K Nishant, P Sharma, V Krishna, C Gupta, KP Singh, N Nitin, R Rastogi. Load balancing of nodes in cloud using ant colony optimization, in *Computer Modelling and Simulation (UKSim), 2012 UKSim 14th International Conference on,* (IEEE, 2012), pp. 3–8
21. Z Zhang, X Zhang. A load balancing mechanism based on ant colony and complex network theory in open cloud computing federation, in *Industrial Mechatronics and Automation (ICIMA), 2010 2nd International Conference on*, vol. 2 (IEEE, 2010), pp. 240–243
22. L Kolb, A Thor, E Rahm. Load balancing for mapreduce-based entity resolution, in *Data Engineering (ICDE), 2012 IEEE 28th International Conference on,* (IEEE, 2012), pp. 618–629
23. J Al-Jaroodi, N Mohamed. DDFTP: dual-direction ftp, in *Proceedings of the 2011 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing,* (IEEE Computer Society, 2011), pp. 504–513
24. AM Nakai, E Madeira, LE Buzato. Load balancing for internet distributed services using limited redirection rates, in *Dependable Computing (LADC), 2011 5th Latin-American Symposium on,* (IEEE, 2011), pp. 156–165
25. SC Wang, KQ Yan, WP Liao, SS Wang. Towards a load balancing in a three-level cloud computing network, in *Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on*, vol. 1 (IEEE, 2010), pp. 108–113
26. H Mehta, P Kanungo, M Chandwani. Decentralized content aware load balancing algorithm for distributed computing environments, in *Proceedings of the International Conference & Workshop on Emerging Trends in Technology,* (ACM, 2011), pp. 370–375
27. P Venkata Krishna, Honey bee behavior inspired load balancing of tasks in cloud computing environments. Appl. Soft Comput. **13**(5), 2292–2303 (2013)
28. RW Clay, NR Wild, DJ Bird, BR Dawson, M Johnston, R Patrick, A Sewell, A cloud monitoring system for remote sites. Publ. Astron. Soc. Aust. **15**(03), 332–335 (1998)
29. J Shao, H Wei, Q Wang, H Mei. A runtime model based monitoring approach for cloud, in *Cloud Computing (CLOUD), 2010 IEEE 3rd International Conference on,* (IEEE, 2010), pp. 313–320
30. E Gelenbe, R Lent. Trade-offs between energy and quality of service, in *Sustainable Internet and ICT for Sustainability (SustainIT),* (IEEE, 2012), pp. 1–5
31. E Gelenbe. Energy packet networks: smart electricity storage to meet surges in demand, in *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering)*, (2012), pp. 1–7
32. E Gelenbe, C Morfopoulou, A framework for energy-aware routing in packet networks. Comput. J. **54**(6), 850–859 (2011)
33. A Berl, E Gelenbe, M Di Girolamo, G Giuliani, H De Meer, MQ Dang, Pentikousis K, Energy-efficient cloud computing. Comput. J. **53**(7), 1045–1051 (2010)
34. D Apostolopoulou, G Gross, Guler T, *Optimized ftr portfolio construction based on the identification of congested network elements. Power systems, IEEE transactions on 28.4*, (2013), pp. 4968–4978
35. X Lu, S Gao, E Ben-Elia, R Pothering, Travelers' day-to-day route choice behavior with real-time information in a congested risky network. Math. Popul. Stud. **21**(4), 205–219 (2014)
36. JY Joo, M Ilic. Distributed scheduling of demand resources in a congested network, in *PES General Meeting| Conference & Exposition, 2014 IEEE,* (IEEE, 2014), pp. 1–5